Reconfiguring Metamorphic Robots via SMT: Is It a Viable Way?

Jan Mrázek, Martin Jonáš, and Jiří Barnat

Abstract— **We present a new approach to tackle the problem of lattice-type metamorphic robots reconfiguration. We base our approach on a reduction to satisfiability modulo theory (SMT). Unlike the current state-of-the-art solutions, we consider the spatial limitations of the modules themselves and produce collision-free plans. We give an in-depth description of the reduction and discuss several optimizations for our technique. We also show an experimental evaluation of our approach and list possible future improvements to our technique.**

I. Introduction

Modular self-reconfigurable robotic platforms, such as M-TRAN [18], ATRON [16], SMORES [15], or HyMOD [19], are used to build larger, more complex robots from small uniform robotic modules. The modules themselves are independent robotic units with the ability to connect to each other and to perform limited locomotion. This provides the metamorphic robots with a unique ability of self-reconfiguration. However, to fully leverage the versatility of such a metamorphic robot, a detailed reconfiguration plan, i.e., a sequence of actions to transform the robot from one configuration to another, is needed. We expect from the plan to respect constraints like the physical limits of modules and surrounding environment, as well as to minimize the time and energy needed for reconfiguration. Finding such a plan efficiently, even with profoundly relaxed constraints, has been proven to be a challenge and is still an open problem.

In this paper, we explore the possibility of leveraging SMT solvers to tackle the problem of reconfiguration. In a nutshell, we construct a logic formula that is satisfiable if and only if there exists a valid reconfiguration plan for a given metamorphic robot, and if so, we use the model of the formula as produced by an SMT solver to derive the plan. Our primary goal is to find out whether such an approach is a viable one in terms of practical usability.

There is a variety of work dealing with the reconfiguration problem. The approaches differ mostly in the mathematical model of the underlying metamorphic system. Some of the work considers only the logical arrangement of the modules and ignores their positioning in space. Therefore, the reconfiguration plan is merely a sequence of connect/disconnect operations [13]. There the authors show that the problem of finding the shortest reconfiguration plan between two configurations is NP-complete.

Other results aim at finding a feasible plan rather than the shortest one. To name a few, [6] uses the divide and conquer approach, [2] uses the state-space search with edit distance heuristics. As the reconfiguration on connector graphs is NP-complete, [12] presents a solution to the reconfiguration via reduction to the Boolean satisfiability problem (SAT). Note that since these solutions ignore module positioning in space, they might produce a plan which is not collision-free. Such plans are suitable for robots with good self locomotion, e.g., SMORES [15], [17].

There are also approaches considering module position in space during the reconfiguration. These are usually motivated by the locomotion of the whole metamorphic robot: [5] presents some locomotion primitives for Roombots, [23] presents an algorithm for locomotion through reconfiguration of M-TRANs. [20] presents a reconfiguration to a line using M-BLOCKS considering the position of the modules in space. Another approach is used by [22] — they pre-synthesize individual module locomotion on the surface of a given scaffold and then consider building larger configuration containing only these scaffolds.

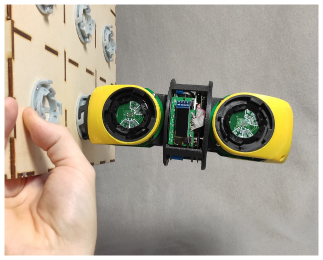
Using SMT for motion planning is not a new approach, see, e.g., [14], [21]. However, these works focus on a rather high-level motion planning of independent robots, while we are aiming for a rather low-level reconfiguration locomotion plan.

II. Satisfiability Modulo Theory

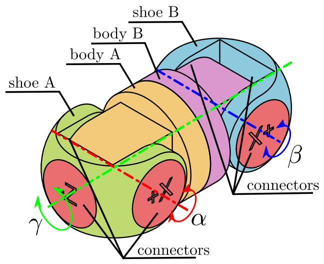
Satisfiability of Boolean formula (SAT) is a problem of finding an assignment of truth values to variables such that a given Boolean formula (e.g., $(x \wedge y) \rightarrow z$) is true [1]. SAT has been successfully applied to many problems. Unfortunately, for many areas, Boolean formulas lack expressiveness. The next logical step is to use first-order logic formulas (e.g., $x + y < 10 \wedge x = y$). However, for practical applications, we are usually not interested in finding a non-standard interpretation of the symbols $+$, $<$ in making the formula satisfiable, we rather interpret the variables as integers and use the standard interpretation of operations. This is where the satisfiability *modulo theory* (SMT) comes in [1].

Not only the introduction of a fixed theory makes the problem more practical, it usually allows for building more efficient tools. The tools can apply algorithms suitable for a given theory. There are several commonly used theories (see [3] for their list). For our work, we are interested in the theory of non-linear real arithmetic — basically, first-order formulas on real-valued variables featuring standard operations like addition or multiplication. Compared to the theory of linear arithmetic, the formula can contain the multiplication of arbitrary expressions [1]. This theory is

J. Mrázek and J. Barnat are with Faculty of Informatics, Masaryk University, Brno, Czech Republic, `jan.mrazek@mail.muni.cz`, `barnat@fi.muni.cz`. M. Jonáš collaborated on the research while he had been PhD student at Faculty of Informatics, Masaryk University.



(a) RoFI module photo.



(b) The module schematics.

Fig. 1: The universal module of the RoFI platform.

usually denoted as NRA (or QF_NRA for its quantifier-free fragment).

III. Terminology & module model

We demonstrate our approach on RoFI modules – see Figure 1a. RoFI arrangement covers many existing robots and, therefore, serves best to show the reduction principle (see Section IV). Note that the adaptation of our approach to another module shape or non-metamorphic modules is straightforward and does not compromise the main idea.

A. RoFI Module

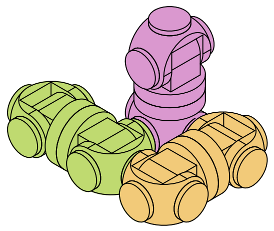
RoFI module has two bodies that rotate around the γ-axis (see Figure 1b). Each body has another spherical body (further referred to as *shoe*) attached to it, which can rotate around the axes α and β. There are three connectors on each shoe labeled $X+$, $X-$ and $Z-$. For simplicity, we assume the shoes have a diameter one, and therefore, the whole module nicely fits into a unit grid. The coordinate origin of the module is placed in the center of shoe A. When two modules connect via a pair of connectors, they can do so in four different orientations rotated by 90°.

B. Configuration

A *configuration* (denoted as c) is a set of *modules* (denoted as M) together with their *connections* (binary relation on connectors). See Figure 2 for a simple example. For our purposes, we consider modules to be distinguishable (i.e., each module has its identifier). We consider two different representations to define the position of modules in space: the *internal* and the *external* one. The internal representation assigns a position to each of the modules' joints; hence it does not directly describe the position of individual modules in the space. On the other hand, the external representation assigns a spatial position and orientation to each shoe of each module, leaving out any information about the rotation of individual joints. Note that both representations have the same expressive power and can be computed from the other.

C. Reconfiguration

The *reconfiguration plan* for two configurations $c_{initial}$ and c_{target} is a sequence $(c_{initial}, c_2, \cdots, c_{n-1}, c_{target})$ of configurations such that: All configurations are strongly connected (as we model modules without locomotion). And for every two consecutive configurations c_i and c_{i+1} it holds that either they are the same or c_{i+1} can be obtained from c_i by applying exactly one of the following actions: releasing some connections, establishment of new connections, or joints' movement.



$$M = \{M_g, M_y, M_p\}$$
$$C = \{conn_{M_g,B,Z-,M_y,X-,0}, \\ conn_{M_y,B,X-,M_p,X+,90}\}$$
$$M_g(\alpha,\beta,\gamma) = (0°, 0°, 0°)$$
$$M_y(\alpha,\beta,\gamma) = (0°, 0°, 0°)$$
$$M_p(\alpha,\beta,\gamma) = (-90°, 0°, 0°)$$

Fig. 2: Example of a configuration in the internal representation.

IV. Reconfiguration using SMT

Let us assume for now that we can construct a first-order formula $\pi_n(c_{initial}, c_{target})$[1] in the theory of non-linear real arithmetic such that π_n is satisfiable if and only if there exists a valid reconfiguration plan of length n from $c_{initial}$ to c_{target}. Then we can pass π_n to an appropriate solver (Z3 [8], SMTRAT [7], dReal [11], Yices2 [9], CVC4 [4]) and find out whether it is satisfiable or not. If so, we can easily extract the reconfiguration plan from the model of the formula. Note that this approach is similar to bounded model checking of programs.

To find the shortest path, we may iteratively construct π_2, π_3, π_4, $\cdots$ to find out the first satisfiable formula. A possible improvement of this naive approach would be either to iteratively extend the current formula and leverage solver caching or to guess an upper bound on the length of the shortest reconfiguration plan and use binary search to find the lowest n for which π_n is satisfiable. The challenge remaining is the construction of π_n, which we describe below.

A. How to Represent a Configuration

Each configuration is represented by a vector $\bar{c}$ of real and Boolean variables representing the positions and rotations of all modules and their connections. Namely, we introduce the following variables.

- For external description:
  - For each shoe A of each module: variables $A_x, A_y, A_z \in \mathbb{R}$ representing the position of the shoe in space, and variables A_{qa}, A_{qb}, A_{qc} and A_{qd}, which represent its orientation in space using a quaternion;
  - for every two shoes A and B: a Boolean variable $conn_{A,B} \in \mathbb{B}$ representing whether the shoes are connected (this may not be necessary, as it can be easily inferred from the internal description of connections, which is described below).

  Note that we use quaternions to capture the orientation rather than Euler angles, as their usage yields a simpler formula.

[1]Further in the text, we omit the arguments $c_{initial}$ and c_{target} if they are clear from the context.

- For internal description:
  - for each module M: variables $M_\alpha, M_\beta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, and $M_\gamma \in [-\pi, \pi]$, which describe the state of the three joints of the module,
  - for every two connectors a, b of different shoes A, B, respectively, and an orientation $r \in \{-\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$: a Boolean variable $conn_{A,a,B,b,r} \in \mathbb{B}$ representing whether the connectors are connected with the given rotation.

We use both internal and external representation in the definition of formula as some properties of the configuration are more straightforward to express in one representation rather than in the other.

B. Intuitive Meaning of The Formulas

We define the main formula π_n expressing the existence of a reconfiguration plan of length n in terms of individual, simple subformulas. Here we give their list together with their intuitive meaning.

- $\varphi_{valid}(\overline{c})$ – the assignment to $\overline{c}$ represents a valid configuration,
- $\varphi_{step}(\overline{c}, \overline{c}')$ – the configuration $\overline{c}$ can be transformed to $\overline{c}'$ by one action (we consider multiple simultaneous connections/disconnections as one action),
- $\varphi_{consis}(\overline{c})$ – the external and internal descriptions represented by $\overline{c}$ are consistent,
- $\varphi_{noIntersect}(\overline{c})$ – no two modules of $\overline{c}$ intersect each other,
- $\varphi_{isConnected}(\overline{c})$ – the graph induced by the configuration is connected (the modules form a single robot),
- $\varphi_c(\overline{c}, \overline{c}')$ configuration $\overline{c}'$ can be obtained from $\overline{c}$ by arbitrary many connections,
- $\varphi_d(\overline{c}, \overline{c}')$ configuration $\overline{c}'$ can be obtained from $\overline{c}$ by arbitrarily many disconnections,
- $\varphi_r(\overline{c}, \overline{c}')$ configuration $\overline{c}'$ can be obtained from $\overline{c}$ by arbitrarily many rotations (for now, by multiples of $\frac{\pi}{2}$),
- $\varphi_{shoeConsistent}(M)$ the shoes of the module M are oriented and placed in space such that they respect the position of module joints,
- $\varphi_{conConsistent}(M, N)$ the shoes of the modules M and N are oriented and placed in space such that they respect any connection between them.

C. Definitions of The Formulas:

- A path of length n exists if there is a sequence of n *valid* configurations starting with the initial configuration, terminating with the target configuration, and it is possible to step between every two successive configurations:

$$\pi_n \overset{\text{def}}{=} \exists \overline{c_1}, \overline{c_2}, \dots, \overline{c_n} \Big(\overline{c_1} = \overline{c_{start}} \quad \land \quad \overline{c_n} = \overline{c_{end}} \quad \land \quad \bigwedge_{1 \le i \le n} \varphi_{valid}(\overline{c_i}) \quad \land \bigwedge_{1 \le i < n} \varphi_{step}(\overline{c_i}, \overline{c_{i+1}}) \Big)$$

- The configuration is valid precisely if its external and internal descriptions are consistent, no two of its shoes intersect each other, all pairs of connected shoes are aligned, and the configuration is connected.

$$\varphi_{valid}(\overline{c}) \overset{\text{def}}{=} \varphi_{consis}(\overline{c}) \land \varphi_{noIntersect}(\overline{c}) \land \varphi_{isConnected}(\overline{c})$$

- The configuration $\overline{c}'$ can be obtained from $\overline{c}$ by one action precisely if connections, disconnections, or rotation can produce it.

$$\varphi_{step}(\overline{c}, \overline{c}') \overset{\text{def}}{=} \varphi_c(\overline{c}, \overline{c}') \lor \varphi_d(\overline{c}, \overline{c}') \lor \varphi_r(\overline{c}, \overline{c}')$$

- External and internal descriptions represented by $\overline{c}$ are consistent precisely if the position in space and orientation of shoes of each module corresponds to the joint positions, and the position and orientation of each two connected modules agree.

$$\varphi_{consis}(\overline{c}) \overset{\text{def}}{=} \bigwedge_{A,B \in \text{modules}(\overline{c})} \varphi_{shoeConsistent}(A, B) \qquad \bigwedge_{m,n \in \text{modules}(\overline{c}) \times \text{modules}(\overline{c})} \varphi_{conConsistent}(m, n)$$

- No two shoes of a configuration intersect themselves precisely if the distance of each two shoes is at least 1 (where 1 is the diameter of one shoe). In other words, to avoid the need to compute square roots, the square of the distance is at least 1:

$$\varphi_{noIntersect}(\overline{c}) \overset{\text{def}}{=} \bigwedge_{A,B \in \text{modules}(\overline{c})} \Big((A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2 \ge 1 \Big)$$

- The configuration $\overline{c}$ is connected precisely if every module is reachable by the connections in up to n steps. See [10] for precise formulation and definition of $\varphi_{isConnected}$.
- The configuration $\overline{c}'$ can be obtained from $\overline{c}$ by connections precisely if the two configurations have exactly the same joint positions and all connections of $\overline{c}$ are also present in $\overline{c}'$.

$$\varphi_c(\overline{c}, \overline{c}') \overset{\text{def}}{=} \bigwedge_{M_\omega \in \text{joints}(\overline{c})} (M_\omega = M'_\omega) \land \bigwedge_{A,a,B,b,r \in \text{connections}(\overline{c})} \big(conn_{A,a,B,b,r} \to conn'_{A,a,B,b,r} \big)$$

- The configuration $\overline{c}'$ can be obtained from $\overline{c}$ by disconnections precisely if $\overline{c}$ can be obtained from $\overline{c'}$ by connections.

$$\varphi_d(\overline{c}, \overline{c}') \overset{\text{def}}{=} \varphi_c(\overline{c}', \overline{c})$$

- A configuration $\overline{c}'$ can be obtained from $\overline{c}$ by rotations precisely if no connection was released nor established and if every intermediate joint position is valid.

$$\varphi_r(\overline{c}, \overline{c}') \overset{\text{def}}{=} \bigwedge_{A,a,B,b,r \in \text{connections}(\overline{c})} (conn_{A,a,B,b,r} \leftrightarrow conn'_{A,a,B,b,r}) \land \varphi'_r(\overline{c}, \overline{c}')$$

where:

$$\varphi'_r(\overline{c}, \overline{c}') \overset{\text{def}}{=} \forall \alpha''_1, \alpha''_2, \cdots, \alpha''_n . \bigwedge_{\alpha_i \in \text{joints}(\overline{c})} (\alpha_i \le \alpha''_i \le \alpha'_i \lor \alpha''_i \le \alpha''_i \le \alpha_i) \to \varphi_{valid}(\overline{c}[\alpha_1 = \alpha''_1, \cdots, \alpha_n = \alpha''_n])$$

- The shoes of the modules are consistent precisely if their position and orientation agree with the transformation depicted in Figure 3. We represent the orientation using quaternions as they lead to a shorter formula with fewer multiplications and trigonometric functions compared to representing the orientation using Euler angles. Quaternions, however, come at the cost of slightly complicated equality – there exist two quaternions representing the same orientation.

$$\begin{aligned}\varphi_{shoeConsistent} A, B \overset{\text{def}}{=} \\ &\varphi_{tPos}(A_x, A_y, A_z) \wedge \\ &\varphi_{tRot}(A_{qa}, A_{qb}, A_{qc}, A_{qd}) \wedge \\ &A'_x = B_x \wedge A'_y = B_y \wedge A'_z = B_z \wedge \\ &\varphi_{quatEq}(A'_{qa}, A'_{qb}, A'_{qc}, A'_{qd}, \\ &\qquad B'_{qB}, B'_{qb}, B'_{qc}, B'_{qd})\end{aligned}$$

where:

$$\begin{aligned}\varphi_{quatEq}(A_{qa}, A_{qb}, A_{qc}, A_{qd}, B_{qB}, B_{qb}, B_{qc}, B_{qd}) \overset{\text{def}}{=} \\ &(A_{qa} = B_{qa} \wedge A_{qb} = B_{qb} \wedge \\ &\ A_{qc} = B_{qc} \wedge A_{qd} = B_{qd}) \vee \\ &(A_{qa} = -B_{qa} \wedge A_{qb} = -B_{qb} \wedge \\ &\ A_{qc} = -B_{qc} \wedge A_{qd} = -B_{qd})\end{aligned}$$

$$\begin{aligned}\varphi_{tPos}(A_x, A_y, A_z) \overset{\text{def}}{=} \\ &A'_x = A_x + 2\,(A_{qa}A_{qc} + A_{qb}A_{qd})\cos(\alpha) + \ \wedge \\ &\qquad 2\,(A_{qa}A_{qd} - A_{qb}A_{qc})\sin(\alpha) \\ &A'_y = \cdots \wedge A'_z = \cdots\end{aligned}$$

$$\begin{aligned}\varphi_{tRot}(A_x, A_y, A_z) \overset{\text{def}}{=} \\ &A'_{qa} = A_{qa}\sin\left(\frac{a_\gamma}{2}\right)\sin\left(\frac{a_\alpha}{2} + \frac{a_\beta}{2}\right) + \ \wedge \\ &\qquad A_{qb}\sin\left(\frac{a_\gamma}{2}\right)\cos\left(\frac{a_\alpha}{2} + \frac{a_\beta}{2}\right) - \\ &\qquad A_{qc}\cos\left(\frac{a_\gamma}{2}\right)\cos\left(\frac{a_\alpha}{2} - \frac{a_\beta}{2}\right) - \\ &\qquad A_{qd}\sin\left(\frac{a_\alpha}{2} - \frac{a_\beta}{2}\right)\cos\left(\frac{a_\gamma}{2}\right) \\ &A'_{qb} = \cdots \wedge A'_{qc} = \cdots \wedge A'_{qd} = \cdots\end{aligned}$$

Note that we omit similar and generic parts of the formula for the sake of simplicity.

- Shoes of different modules are consistent precisely if there is no connection between them or if there is a connection between them and they obey the coordinate transformation shown in Figure 3.

$$\begin{aligned}\varphi_{conConsistent}(A, B) \overset{\text{def}}{=} \bigwedge_{(a,b,r)\in\text{all connector combinations}} conn_{A,a,B,b,r} \rightarrow \\ &\varphi_{tPos_{a,b,r}}(A_x, A_y, A_z) \wedge \\ &\varphi_{tRot_{a,b,r}}(A_{qa}, A_{qb}, A_{qc}, A_{qd}) \wedge \\ &A'_x = B_x \wedge A'_y = B_y \wedge A'_z = B_z \wedge \\ &\varphi_{quatEq}(A'_{qa}, A'_{qb}, A'_{qc}, A'_{qd}, \\ &\qquad B'_{qB}, B'_{qb}, B'_{qc}, B'_{qd})\end{aligned}$$

We omit the definitions of φ_{tPos} and φ_{tRot} as they are structurally similar to the previous case and only differ in the transformation they define. Also, there are 36 of them.

D. Technical Details

Most of the state-of-the-art SMT solvers for real non-linear arithmetic cannot process a formula containing uni-

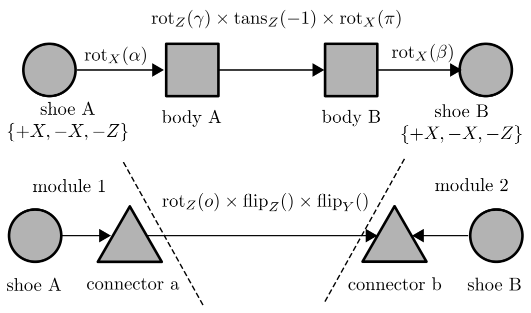


Fig. 3: Illustration of the coordinate transformation between two shoes of a single module and two shoes of different modules

versal quantification nor trigonometric functions. The only exception from the former known to us that can also handle quantification is the solver dReal [11]. Therefore, the reduction we have given above cannot be directly used with the majority of the solvers.

There is no straightforward[2] way to remove the universal quantification in φ_r without an approximation of the solution. However, checking for intermediate position validity in discrete steps is often good enough in practice. The discretization can be easily implemented by limiting maximal joint position change between steps or by fixing the joint step value to a fixed value. Note that this approximation changes the length of the shortest reconfiguration plan as one step with long movement has to be expressed as a series of shorter movements. The definition of φ_r for 90° steps is:

$$\varphi_r(\overline{c}, \overline{c}') \overset{\text{def}}{=} \bigwedge_{A,a,B,b,r\in\text{connections}(\overline{c})} \left(conn_{A,a,B,b,r} \leftrightarrow conn'_{A,a,B,b,r}\right) \wedge$$
$$\bigwedge_{\alpha_i\in\text{joints}(\overline{c})} |\alpha_i - \alpha'_i| = 0 \vee |\alpha_i - \alpha'_i| = \frac{\pi}{2}$$

The reduction we described above uses the functions sine and cosine to express the relation between internal and external configuration. Fortunately, the formula does not contain free-standing appearances of trigonometric function arguments. By using trigonometric equality for sums, we can rewrite the formula such that it contains only expressions in form $\sin\alpha$, $\sin\frac{\alpha}{2}$, and similarly for cosine. Hence, to remove these expressions from the formula, we introduce four new variables: $\alpha_{sin}, \alpha_{sinhalf}, \alpha_{cos}$, and $\alpha_{coshalf}$ which we substitute for the corresponding expressions. Then for every argument α, we add $\varphi_{sinCos}(\alpha)$ (containing well-known trigonometric identities) to π_n:

$$\begin{aligned}\varphi_{sinCos}(\alpha) \overset{\text{def}}{=} &\ \alpha_{sin}^2 + \alpha_{cos}^2 = 1 \\ &\wedge \alpha_{sin} = 2\cdot\alpha_{sinhalf}\cdot\alpha_{coshalf} \\ &\wedge \alpha_{cos} = 1 - 2\cdot\alpha_{sinhalf}^2\end{aligned}$$

Note that in a similar way, we have to rewrite the alternative definition of φ_r using only function values and not the joints' positions directly.

Given our model of the robots, we see that every two shoes can have at most one connection, and each connector can

[2]It can be done by cylindrical algebraic decomposition

be connected at most once. Therefore, in our configuration representation, at most one of the variables $conn_{A,a,B,b,r}$ for given A and B can be true. Looking at the definition of φ_{valid}, one can note that the formula implies this. However, many usages of SAT or SMT solvers show that putting additional, more explicit constraints can lead to a speed-up in testing satisfiability. Therefore, we include such explicit constraints in our experiments.

Similarly, we add the possibility to root the first module as a possible optimization: i.e., put its shoe A at $(0, 0, 0)$ with no rotation. Since we do not consider the surrounding environment in our case, the rooting of the first module has no effect on the shortest reconfiguration plan. Rooting the module removes a large number of degrees of freedom from the formula.

V. Experimental Evaluation

To evaluate our work, we implemented the reduction in C++ using the API of Z3. Our tool[3] can either find the shortest path using the algorithm from Section IV or construct the formula π_n for all n up to the upper bound given as input and output them to the SMT-LIB2 format [3]. It is also possible to include the optimizations presented in Subsection IV-D. To ensure the correctness of our implementation, we wrote several unit tests testing the subformulas on hand-crafted inputs. The inputs were based partially on experiments presented by [2]. We also included several simple and several challenging tasks with up to 10 modules.

We chose several SMT solvers based on the results from SMT-Competition[4]. We excluded the solver dReal as it runs out of memory on all our test cases. We compiled several versions of the solvers and decided the satisfiability of formulas produced by our tool on several reconfigurations tasks. For the evaluation, we use a timeout of 2500 seconds and a memory limit of 20 GB on the AMD EPYC 7371 CPU.

When the formula is satisfiable (based on the knowledge of the shortest path), the solvers in more than a third of the cases found a solution. When the formula is unsatisfiable, the solvers could find a counterexample only for a small number of steps and modules. The solvers did not handle formulas without optimizations well. The solving times vary a lot, and we consider them unpredictable.

We also observed that for several cases, the newer solvers performed better. We present a summary Table I of two hand-picked examples of this phenomenon: 3-reattach and 6-roller. These examples are not the best performing, but they show our observations the best. The 3-reattach contains three modules, and the goal is to reattach the traveling module from one side of the root module to the other one. The 6-roller transforms the initial blob of six modules into a ring.

The formulas produced by our tool have roughly 18k nodes for two modules and reconfiguration of length two, 49k nodes for three modules, 423k nodes for eight modules. Since the formula size is linear to the number of steps, it is easy to extrapolate. The formula size grows to 58k, 212k, and 4603k, respectively, when we append the constraints explicitly limiting the number of connections per connector.

When considering the performance of the whole process, we cannot directly and precisely compare our measurement with other state-of-the-art tools. Not only do they use a slightly different model of a module, but the tools are not available, and their evaluation does not contain any run times.

We also run our benchmarks with the optimizations mentioned in Subsection IV-D. Our hypothesis that rooting a module should help the solver to find the solution more quickly was confirmed. Without this constraint, even simple test cases were not solved by any solver. Adding the explicit constraint on the number of connections of a single connector dramatically increases the size of the formula. However, for the Z3 solver, it drastically improved the performance. For the other solvers, the performance was slightly increased for small instances; for the others, it made the performance much worse. Z3 offers a non-standard extension to the SMT-LIB language, a command to specify the maximum number of variables out of a given set, which can be true simultaneously. We suspect that Z3 recognizes our construction and replaces it with this command internally.

Unfortunately, we see the performance of our approach as insufficient for direct deployment on a system of metamorphic robots. However, we note a positive trend in our evaluation — the newer versions of the solvers perform much better than the older ones on some test cases. Therefore, we hope that with advances in SMT solving, our method without a change could perform much better in the future.

VI. Possible Improvements

We think our method can scale better in the future, we do not consider it perfect, and we are looking for some improvements.

At first, we can observe that the SMT solvers are far from being optimized to the type of formulas we produce in our approach. Hence, getting closer to the SMT community and providing the community with the formulas we need to solve may significantly improve the performance of SMT solvers in our case in the future.

We also performed experiments with omitting constraints from the formula. We noted that especially when omitting shoe and connector consistency, the solvers perform much better. However, and predictably, they produce infeasible reconfiguration plans. This observation leads us to implement a counterexample guided refinement. The general idea would be to check for the satisfiability of a simple formula describing the reconfiguration without many constraints. If the plan is feasible, we found a solution. Otherwise, the conflicting actions are forbidden in the formula by adding some more constraints, and the process is restarted. However, for this approach to scale well, it is necessary to have a way of generalizing the concrete counterexamples to more general ones. The generalization is what we currently struggle with.

[3]Available at `https://github.com/paradise-fi/RoFI/tree/master/softwareComponents/smtreconfig`

[4]https://smt-comp.github.io/

TABLE I: Table depicting the ratio of successfully solved formulas and the average solving time for given versions of solvers. The additional constraints are: rooted module (R) and limit the number of connections per connector (C).

Test case	Constraints	cvc4-1.5	cvc4-1.7	cvc4-up	yices-2.4.0	yices-2.5.0	yices-2.6.1	yices-up	z3-4.4.0	z3-4.5.0	z3-4.7.1	z3-4.8.1	z3-up
3-attach	–	18 % 223 s	27 % 385 s	27 % 446 s	9 % 0 s	9 % 0 s	9 % 0 s	9 % 0 s	9 % 0 s	18 % 1680 s	9 % 0 s	9 % 0 s	9 % 0 s
	R	27 % 5 s	27 % 3 s	27 % 3 s	9 % 0 s	36 % 26 s	36 % 22 s	45 % 28 s	27 % 80 s	36 % 43 s	36 % 27 s	36 % 26 s	27 % 10 s
	R C	27 % 10 s	27 % 10 s	27 % 15 s	9 % 0 s	18 % 73 s	18 % 50 s	27 % 60 s	27 % 29 s	27 % 50 s	36 % 39 s	54 % 24 s	27 % 11 s
6-roller	–	0 % –	0 % –	0 % –	0 % –	0 % –	0 % –	9 % 0 s	9 % 283 s	0 % –	0 % –	0 % –	0 % –
	R	9 % 391 s	9 % 384 s	9 % 393 s	0 % –	0 % –	0 % –	0 % –	36 % 1400 s	18 % 716 s	9 % 62 s	9 % 22 s	0 % –
	R C	9 % 835 s	9 % 640 s	9 % 687 s	0 % –	0 % –	0 % –	0 % –	36 % 415 s	45 % 2388 s	54 % 38 s	81 % 69 s	36 % 28 s

At last, it might be worth it to explore possible optimization of our configuration representation, possibly to even come up with a different representation, which would simplify the computationally most challenging parts of the formula – checking $\varphi_{shoeConsistent}$ and $\varphi_{conConsistent}$. If we sacrifice a class of reconfiguration problems requiring non $90°$ rotations, we could leverage a grid-based reconfiguration.

VII. Conclusions

We have proposed a novel solution for the reconfiguration of modular metamorphic robots using the reduction to SMT. Unlike the other work solving similar problems via reduction, our solution provides collision-free plans. Nevertheless, we conclude that the direct reduction of the reconfiguration problem to SMT is not much viable yet. Furthermore, we also conclude that the speed with which the solvers are improved does not indicate that the situation will significantly change in the near future, though the progress in the efficiency of SMT solvers is undeniable.

VIII. Acknowledgment

This research was supported by ERDF ”CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence” (No. CZ.02.1.01/0.0/0.0/16 019/0000822).

References

[1] *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[2] M. Asadpour, M. H. Z. Ashtiani, A. Spröwitz, and A. J. Ijspeert. Graph signature for self-reconfiguration planning of modules with symmetry. In *International Conference on Intelligent Robots and Systems*. IEEE, 2009.

[3] C. Barrett, P. Fontaine, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[4] C. W. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Computer Aided Verification*, Lecture Notes in Computer Science. Springer, 2011.

[5] S. Bonardi, R. Moeckel, A. Sproewitz, M. Vespignani, and A. J. Ijspeert. Locomotion through reconfiguration based on motor primitives for roombots self-reconfigurable modular robots. In *ROBOTIK 2012*. VDE-Verlag, 2012.

[6] A. Casal and M. H. Yim. Self-reconfiguration planning for a class of modular robots. In *Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839. International Society for Optics and Photonics, SPIE, 1999.

[7] F. Corzilius, G. Kremer, S. Junges, S. Schupp, and E. Ábrahám. SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In *Theory and Applications of Satisfiability Testing*, Lecture Notes in Computer Science. Springer, 2015.

[8] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*. Springer, 2008.

[9] B. Dutertre. Yices 2.2. In *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.

[10] Y. Filmus. SAT algorithm for determining if a graph is disjoint.

[11] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *Automated Deduction – CADE-24*, volume 7898 of *Lecture Notes in Computer Science*. Springer, 2013.

[12] A. A. Gorbenko and V. Y. Popov. Programming for modular reconfigurable robots. *Programming and Computer Software*, 38(1), 2012.

[13] F. Hou and W. Shen. Graph-based optimal reconfiguration planning for self-reconfigurable robots. *Robotics and Autonomous Systems*, 62(7), 2014.

[14] F. Imeson and S. L. Smith. An SMT-Based Approach to Motion Planning for Multiple Robots with Complex Constraints. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2019.

[15] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit. An End-To-End System for Accomplishing Tasks with Modular Robots. In *Robotics: Science and Systems XII*, 2016.

[16] M. W. Jörgensen, E. H. Östergaard, and H. H. Lund. Modular ATRON: modules for a self-reconfigurable robot. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2004.

[17] C. Liu, M. Whitzer, and M. Yim. A Distributed Reconfiguration Planning Algorithm for Modular Robots. *IEEE Robotics and Automation Letters*, 4(4), 2019.

[18] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME transactions on mechatronics*, 7(4), 2002.

[19] C. Parrott, T. J. Dodd, and R. Gross. HyMod: A 3-DOF Hybrid Mobile and Self-Reconfigurable Modular Robot and its Extensions. In *Distributed Autonomous Robotic Systems*, volume 6 of *Springer Proceedings in Advanced Robotics*. Springer, 2016.

[20] J. Romanishin, J. Mamish, and D. Rus. Decentralized Control for 3D M-Blocks for Path Following, Line Formation, and Light Gradient Aggregation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2019.

[21] P. Surynek. Lazy Compilation of Variants of Multi-robot Path Planning with Satisfiability Modulo Theory (SMT) Approach. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2019.

[22] P. Thalamy, B. Piranda, F. Lassabe, and J. Bourgeois. Scaffold-based Asynchronous Distributed Self-Reconfiguration by Continuous Module Flow. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2019.

[23] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. A Self-Reconfigurable Modular Robot: Reconfiguration Planning and Experiments. *International Journal of Robotic Research – IJRR*, 21, 2002.